Пензенский государственный университет

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Отчет о лабораторной работе №6

по дисциплине «Разработка кроссплатформенных приложений»

на тему «Сетевое взаимодействие в Java»

Выполнили: ст-ты гр. 22ВОЭ1

Улитина А.В.

Нагаев Д.А.

Проверили: к.т.н., доцент каф. ВТ

Юрова О.В.

Карамышева Н.С.

2025

Цель работы

Научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

Задание

Модифицировать приложение из предыдущей лабораторной работы, реализовав клиент-серверную архитектуру, обеспечивающую распределенное вычисление определенного интеграла на нескольких вычислительных узлах (клиентах) при этом каждый узел использует несколько нитей, как в предыдущей работе. Сервер не занимается вычислениями, а лишь реализует взаимодействие с пользователем и агрегацию результатов вычислений от клиентов. Нечетные варианты используют протокол UDP.

Ход работы

Листинг программы

import javax.swing.JOptionPane;

import javax.swing.\*;

import java.net.\*;

import java.io.\*;

import javax.swing.table.DefaultTableModel;

import java.util.ArrayList;

import java.util.concurrent.atomic.AtomicReference;

public class NewJFrame extends javax.swing.JFrame {

private DatagramSocket socket;

private InetAddress serverAddress;

private static final int SERVER\_PORT = 9876;

private ArrayList<RecIntegral> integralsList;

public NewJFrame() {

initComponents();

integralsList = new ArrayList<>();

try {

socket = new DatagramSocket();

serverAddress = InetAddress.getByName("localhost"); // или IP сервера

// Отправляем серверу уведомление о подключении

String connectMessage = "CLIENT\_CONNECTED";

byte[] connectData = connectMessage.getBytes();

DatagramPacket connectPacket = new DatagramPacket(

connectData, connectData.length,

serverAddress, SERVER\_PORT

);

socket.send(connectPacket);

// Ждем подтверждение (опционально)

byte[] ackBuffer = new byte[1025];

DatagramPacket ackPacket = new DatagramPacket(ackBuffer, ackBuffer.length);

socket.receive(ackPacket);

String ack = new String(ackPacket.getData(), 0, ackPacket.getLength());

System.out.println("подключено к серверу.");

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

public void Table(){

}

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jButton1 = new javax.swing.JButton();

jScrollPane1 = new javax.swing.JScrollPane();

jTable1 = new javax.swing.JTable();

jButton2 = new javax.swing.JButton();

jButton3 = new javax.swing.JButton();

jTextFieldLowLim = new javax.swing.JTextField();

jTextFieldUpLim = new javax.swing.JTextField();

jTextFieldStep = new javax.swing.JTextField();

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

jLabel3 = new javax.swing.JLabel();

jButton4 = new javax.swing.JButton();

jButton5 = new javax.swing.JButton();

jLabel4 = new javax.swing.JLabel();

jLabel5 = new javax.swing.JLabel();

jLabel6 = new javax.swing.JLabel();

jButton6 = new javax.swing.JButton();

jButton7 = new javax.swing.JButton();

jButton8 = new javax.swing.JButton();

jButton9 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jButton1.setText("Удалить");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jTable1.setModel(new javax.swing.table.DefaultTableModel(

new Object [][] {

},

new String [] {

"Нижняя граница", "Верхняя гранциа", "Шаг", "Результат"

}

) {

boolean[] canEdit = new boolean [] {

false, false, false, false

};

public boolean isCellEditable(int rowIndex, int columnIndex) {

return canEdit [columnIndex];

}

});

jScrollPane1.setViewportView(jTable1);

jButton2.setText("Рассчитать");

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

jButton3.setText("Добавить");

jButton3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton3ActionPerformed(evt);

}

});

jTextFieldLowLim.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jTextFieldLowLimActionPerformed(evt);

}

});

jLabel1.setText("Нижняя граница");

jLabel2.setText("Верхняя граница");

jLabel3.setText("Шаг интегрирования");

jButton4.setText("Заполнить");

jButton4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton4ActionPerformed(evt);

}

});

jButton5.setText("Очистить");

jButton5.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton5ActionPerformed(evt);

}

});

jLabel4.setText("Работа со списком");

jLabel5.setText("Работа с текстом");

jLabel6.setText("Работа с двоичным кодом");

jButton6.setText("Загрузка");

jButton6.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton6ActionPerformed(evt);

}

});

jButton7.setText("Сохранение");

jButton7.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton7ActionPerformed(evt);

}

});

jButton8.setText("Загрузка");

jButton8.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton8ActionPerformed(evt);

}

});

jButton9.setText("Сохранение");

jButton9.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton9ActionPerformed(evt);

}

});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(28, 28, 28)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 441, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(44, 44, 44)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jButton2)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jButton1)

.addComponent(jButton3)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jButton4)

.addComponent(jButton5))

.addGap(8, 8, 8))

.addComponent(jLabel4, javax.swing.GroupLayout.Alignment.TRAILING))

.addGap(24, 24, 24))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()

.addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 86, Short.MAX\_VALUE)

.addComponent(jLabel6)

.addContainerGap())

.addGroup(layout.createSequentialGroup()

.addComponent(jButton6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(jButton8)

.addGap(46, 46, 46))

.addGroup(layout.createSequentialGroup()

.addComponent(jButton7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(jButton9)

.addGap(36, 36, 36))))

.addGroup(layout.createSequentialGroup()

.addGap(41, 41, 41)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1)

.addComponent(jTextFieldLowLim, javax.swing.GroupLayout.PREFERRED\_SIZE, 107, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(40, 40, 40)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel2)

.addComponent(jTextFieldUpLim, javax.swing.GroupLayout.PREFERRED\_SIZE, 118, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(35, 35, 35)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(jTextFieldStep))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addContainerGap(32, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(jLabel2)

.addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jTextFieldLowLim, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jTextFieldUpLim, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jTextFieldStep, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(62, 62, 62)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 288, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(14, 14, 14))

.addGroup(layout.createSequentialGroup()

.addGap(92, 92, 92)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(jButton4)

.addGap(18, 18, 18)

.addComponent(jButton5))

.addGroup(layout.createSequentialGroup()

.addComponent(jButton3)

.addGap(18, 18, 18)

.addComponent(jButton1)

.addGap(18, 18, 18)

.addComponent(jButton2)))

.addGap(38, 38, 38)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel5)

.addComponent(jLabel6))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton6)

.addComponent(jButton8))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton7)

.addComponent(jButton9))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

pack();

}// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

int rowNum = jTable1.getSelectedRow();

if (rowNum == -1){

JOptionPane.showMessageDialog(null, "выберите строку");

}else tModel.removeRow(rowNum);

}

private void jTextFieldLowLimActionPerformed(java.awt.event.ActionEvent evt) {

}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

try {

double upperLim;

double lowLim;

double step;

lowLim = Double.parseDouble(jTextFieldLowLim.getText());

upperLim = Double.parseDouble(jTextFieldUpLim.getText());

step = Double.parseDouble(jTextFieldStep.getText());

if (upperLim < 0.000001 || upperLim > 1000000 ||

lowLim < 0.000001 || lowLim> 1000000 ||

step < 0.000001 || step > 1000000) {

throw new CustomException("Значения должны находиться в диапазоне от 0.000001 до 1000000.");

}

if (upperLim < lowLim){

throw new CustomException("Нижний предел должен быть меньше верхнего предела.");

}

if ((upperLim - lowLim) < step){

throw new CustomException("Шаг интегрирования должен быть меньше интервала.");

}

// Создаем экземпляр RecIntegral с проверкой данных

RecIntegral recIntegral = new RecIntegral(lowLim, upperLim, step, 0.0);

// Добавляем строку в таблицу

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

tModel.addRow(new Object[]{lowLim, upperLim, step});

// Добавляем запись в коллекцию

integralsList.add(recIntegral);

// Очищаем поля ввода

jTextFieldLowLim.setText("");

jTextFieldUpLim.setText("");

jTextFieldStep.setText("");

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Введены некорректные данные. Пожалуйста, введите числа.", "Ошибка", JOptionPane.ERROR\_MESSAGE);

} catch (CustomException ex) {

JOptionPane.showMessageDialog(this, ex.getMessage(), "Ошибка диапазона", JOptionPane.WARNING\_MESSAGE);

}

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

for (int row = 0; row < tModel.getRowCount(); row++) {

final int finalRow = row;

double a = ((Number) tModel.getValueAt(row, 0)).doubleValue();

double b = ((Number) tModel.getValueAt(row, 1)).doubleValue();

double step = ((Number) tModel.getValueAt(row, 2)).doubleValue();

// Разбиваем интервал на 3 части

double interval = (b - a) / 3;

final double[] totalResult = {0.0};

// Создаем 3 потока

Thread[] threads = new Thread[3];

for (int i = 0; i < 3; i++) {

final double start = a + i \* interval;

final double end = (i == 2) ? b : start + interval;

threads[i] = new Thread(() -> {

double partialSum = 0;

for (double x = start; x < end; x += step) {

partialSum += Math.min (step, (b - start));

}

synchronized (totalResult) {

totalResult[0] += partialSum;

}

});

threads[i].start();

}

// Ждем завершения потоков

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

// Обновляем GUI и отправляем на сервер

SwingUtilities.invokeLater(() -> {

tModel.setValueAt(totalResult[0], finalRow, 3);

integralsList.get(finalRow).setResult(totalResult[0]);

sendResultToServer(a, b, step, totalResult[0]);

});

}

}

class IntegralCalculatorThread extends Thread {

private final double a;

private final double b;

private final double step; // Шаг интегрирования

private double partialSum; // Частичная сумма (результат потока)

public IntegralCalculatorThread(double a, double b, double step) {

this.a = a;

this.b = b;

this.step = step;

this.partialSum = 0;

}

@Override

public void run() {

double start, h, sum=0;

start = a;

do{

h= Math.min (step, (b - start));

sum +=h \* (1/ start + 1/(start + h)) / 2;

start += h;

} while ((start)< b);

this.partialSum = sum;

}

public double getPartialSum() {

return partialSum;

}

}

private void sendResultToServer(double a, double b, double step, double result) {

try {

String data = String.format("a=%.2f, b=%.2f, step=%.4f, result=%.6f", a, b, step, result);

byte[] buffer = data.getBytes();

DatagramPacket packet = new DatagramPacket(buffer, buffer.length, serverAddress, SERVER\_PORT);

socket.send(packet);

} catch (Exception e) {

//System.err.println("Ошибка отправки: " + e.getMessage());

}

}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

tModel.setRowCount(0);

for (RecIntegral rec : integralsList) {

tModel.addRow(new Object[]{

rec.getLowerLimit(),

rec.getUpperLimit(),

rec.getStep(),

rec.getResult()

});

}

}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

tModel.setRowCount(0);

integralsList.clear();

}

private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {

JFileChooser fileChooser = new JFileChooser();

if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

saveToTextFile(file);

}

}

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {

JFileChooser fileChooser = new JFileChooser();

if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

loadFromBinaryFile(file);

}

}

private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {

JFileChooser fileChooser = new JFileChooser();

if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

saveToBinaryFile(file);

}

}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {

JFileChooser fileChooser = new JFileChooser();

if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

loadFromTextFile(file);

}

}

public class InvalidDataException extends Exception {

public InvalidDataException(String message) {

super(message);

}

}

private void saveToTextFile(File file) {

try (PrintWriter writer = new PrintWriter(file)) {

for (RecIntegral rec : integralsList) {

writer.println(rec.getLowerLimit() + "," + rec.getUpperLimit() + "," + rec.getStep() + "," + rec.getResult());

}

JOptionPane.showMessageDialog(this, "Данные успешно сохранены в текстовый файл.", "Успех", JOptionPane.INFORMATION\_MESSAGE);

} catch (IOException e) {

JOptionPane.showMessageDialog(this, "Ошибка при сохранении файла: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);

}

}

private void loadFromTextFile(File file) {

try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

integralsList.clear();

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

tModel.setRowCount(0);

String line;

while ((line = reader.readLine()) != null) {

String[] parts = line.split(",");

double lowerLimit = Double.parseDouble(parts[0]);

double upperLimit = Double.parseDouble(parts[1]);

double step = Double.parseDouble(parts[2]);

double result = Double.parseDouble(parts[3]);

RecIntegral rec = new RecIntegral(lowerLimit, upperLimit, step, result);

integralsList.add(rec);

tModel.addRow(new Object[]{lowerLimit, upperLimit, step, result});

}

JOptionPane.showMessageDialog(this, "Данные успешно загружены из текстового файла.", "Успех", JOptionPane.INFORMATION\_MESSAGE);

} catch (IOException | CustomException e) {

JOptionPane.showMessageDialog(this, "Ошибка при загрузке файла: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);

}

}

private void saveToBinaryFile(File file) {

System.out.println("Сохранение данных: " + integralsList);

if (integralsList == null || integralsList.isEmpty()) {

JOptionPane.showMessageDialog(this, "Нет данных для сохранения.", "Ошибка", JOptionPane.ERROR\_MESSAGE);

return;

}

try (FileOutputStream fileOut = new FileOutputStream(file)) {

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(integralsList);

JOptionPane.showMessageDialog(this, "Данные успешно сохранены в двоичный файл.", "Успех", JOptionPane.INFORMATION\_MESSAGE);

} catch (IOException e) {

JOptionPane.showMessageDialog(this, "Ошибка при сохранении файла: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

throw new RuntimeException(ex);

}

//integralsList.removeAll(integralsList);

}

private void loadFromBinaryFile(File file) {

if (!file.exists() || file.length() == 0) {

JOptionPane.showMessageDialog(this, "Файл не существует или пуст.", "Ошибка", JOptionPane.ERROR\_MESSAGE);

return;

}

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {

integralsList = (ArrayList<RecIntegral>) ois.readObject(); // Десериализуем только integralsList

// Обновляем таблицу

DefaultTableModel tModel = (DefaultTableModel) jTable1.getModel();

tModel.setRowCount(0); // Очищаем таблицу

for (RecIntegral rec : integralsList) {

tModel.addRow(new Object[]{

rec.getLowerLimit(),

rec.getUpperLimit(),

rec.getStep(),

rec.getResult()

});

}

JOptionPane.showMessageDialog(this, "Данные успешно загружены из двоичного файла.", "Успех", JOptionPane.INFORMATION\_MESSAGE);

} catch (IOException | ClassNotFoundException e) {

JOptionPane.showMessageDialog(this, "Ошибка при загрузке файла: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);

}

}

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new NewJFrame().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JButton jButton3;

private javax.swing.JButton jButton4;

private javax.swing.JButton jButton5;

private javax.swing.JButton jButton6;

private javax.swing.JButton jButton7;

private javax.swing.JButton jButton8;

private javax.swing.JButton jButton9;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JLabel jLabel4;

private javax.swing.JLabel jLabel5;

private javax.swing.JLabel jLabel6;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTable jTable1;

private javax.swing.JTextField jTextFieldLowLim;

private javax.swing.JTextField jTextFieldStep;

private javax.swing.JTextField jTextFieldUpLim;

// End of variables declaration

}

import java.io.Serializable;

public class RecIntegral implements Serializable {

private static final long serialVersionUID = 1L;

private double lowerLimit;

private double upperLimit;

private double step;

private double result;

public RecIntegral(double lowerLimit, double upperLimit, double step, double result) throws CustomException {

if (lowerLimit < 0.000001 || lowerLimit > 1000000) {

throw new CustomException ("Значение должно быть в диапазоне от 0.000001 до 1000000.");

}

if (upperLimit < 0.000001 || upperLimit> 1000000) {

throw new CustomException ("Значение должно быть в диапазоне от 0.000001 до 1000000.");

}

if (step < 0.000001 || step > 1000000) {

throw new CustomException ("Значение должно быть в диапазоне от 0.000001 до 1000000.");

}

if (lowerLimit> upperLimit) {

throw new CustomException ("Нижний предел должен быть меньше верхнего предела.");

}

if ((upperLimit - lowerLimit) < step) {

throw new CustomException ("Шаг интегрирования должен быть меньше интервала.");

}

this.lowerLimit = lowerLimit;

this.upperLimit = upperLimit;

this.step = step;

this.result = result;

}

public double getLowerLimit() {

return lowerLimit;

}

public double getUpperLimit() {

return upperLimit;

}

public double getStep() {

return step;

}

public double getResult() {

return result;

}

public void setResult(double result) {

this.result = result;

}

}

public class CustomException extends Exception {

public CustomException(String message) {

super (message);

}

}

import java.net.\*;

public class Server {

public static void main(String[] args) throws Exception {

DatagramSocket socket = new DatagramSocket(9876);

System.out.println("Сервер запущен и ожидает результатов");

while (true) {

byte[] buffer = new byte[1025];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

if (message.equals("CLIENT\_CONNECTED")) {

String response = "SERVER\_ACK";

byte[] responseData = response.getBytes();

DatagramPacket responsePacket = new DatagramPacket(

responseData, responseData.length,

packet.getAddress(), packet.getPort()

);

socket.send(responsePacket);

} else {

System.out.println("Результат от " + packet.getAddress() + ": " + message);

}

}

}

}

Словесное описание алгоритма

1. Объявляем класс NewJFrame, который наследуется от javax.swing.JFrame.
2. Объявляем два приватных поля: socket типа DatagramSocket, который используется для отправки и получения UDP-пакетов, и serverAddress типа InetAddress, который хранит адрес сервера.
3. Определяем константу SERVER\_PORT, которая будет использоваться для указания порта, на котором сервер будет слушать входящие сообщения.
4. Объявляем список integralsList, который будет хранить объекты типа RecIntegral.
5. Конструктор NewJFrame, который вызывается при создании экземпляра этого класса.
6. Инициализируем integralsList как новый объект ArrayList, чтобы мы могли добавлять в него элементы.
7. В блоке try мы создаем новый экземпляр DatagramSocket, который будет использоваться для отправки и получения UDP-пакетов. Если возникнет ошибка при создании сокета, она будет перехвачена в блоке catch.
8. Здесь мы получаем адрес сервера, к которому мы хотим подключиться.
9. Создаем строку connectMessage, которая будет содержать сообщение о подключении клиента к серверу.
10. Преобразуем строку connectMessage в массив байтов, который будет отправлен через сокет.
11. Создаем объект DatagramPacket, который содержит данные для отправки. Мы передаем массив байтов, его длину, адрес сервера и порт.
12. Отправляем созданный пакет с сообщением о подключении на сервер.
13. Создаем буфер ackBuffer, который будет использоваться для получения подтверждения от сервера.
14. Создаем пакет ackPacket, который будет использоваться для получения данных от сервера.
15. Ожидаем, пока сервер отправит ответ, и получаем его в ackPacket. Этот вызов блокирует выполнение, пока не будет получен пакет.
16. Преобразуем полученные данные из ackPacket в строку ack.
17. Выводим сообщение в консоль, подтверждающее успешное подключение к серверу.
18. Закрываем сокет, так как дальнейшее взаимодействие с сервером не требуется.
19. Обрабатываем любые исключения, которые могут возникнуть в процессе выполнения блока try. Если произойдет ошибка, она будет выведена в консоль.
20. Метод, который вызывается при нажатии на кнопку 1.
21. Получение модели таблицы jTable1 и приведение её к типу DefaultTableModel, чтобы можно было изменять данные в таблице.
22. Получение номера выбранной строки в таблице.
23. Проверка, выбрана ли строка. Если нет, то показать сообщение об ошибке, если строка не выбрана.
24. Если строка выбрана, удалить её из модели таблицы.
25. Метод, который вызывается при нажатии на кнопку 3.
26. Начало блока try, в котором будет выполняться код, который может вызвать исключение.
27. Объявление переменных для верхнего предела, нижнего предела и шага интегрирования.
28. Парсинг текста из текстового поля jTextFieldLowLim в число типа double.
29. Парсинг текста из текстового поля jTextFieldUpLim в число типа double.
30. Парсинг текста из текстового поля jTextFieldStep в число типа double.
31. Проверка, находятся ли введенные значения (верхний предел, нижний предел и шаг) в допустимом диапазоне от 0.000001 до 1000000. Если хотя бы одно из условий истинно, то генерация пользовательского исключения CustomException с сообщением об ошибке, если одно из значений не соответствует требованиям.
32. Проверка, что верхний предел больше нижнего предела. Если это не так, то генерация исключения с сообщением, что нижний предел должен быть меньше верхнего.
33. Проверка, что шаг интегрирования меньше интервала между верхним и нижним пределами. Если это не так, то генерация исключения с сообщением, что шаг должен быть меньше интервала.
34. Создание нового объекта RecIntegral с переданными значениями нижнего предела, верхнего предела и шага. Четвертым параметром передается 0.0.
35. Добавление новой строки в таблицу с введенными значениями нижнего предела, верхнего предела и шага.
36. Добавление созданного объекта recIntegral в список integralsList, чтобы сохранить информацию о интеграле.
37. Очистка текстового поля для нижнего предела.
38. Очистка текстового поля для верхнего предела.
39. Очистка текстового поля для шага интегрирования.
40. Обработка исключения, если введенные данные не могут быть преобразованы в числа
41. Метод, который вызывается при нажатии на кнопку 2.
42. Здесь мы получаем модель таблицы (DefaultTableModel), связанной с jTable1.
43. Запускаем цикл, который проходит по всем строкам таблицы. getRowCount() возвращает количество строк в модели.
44. Сохраняем текущий индекс строки в переменной finalRow. Мы используем final, чтобы сделать переменную доступной внутри анонимного класса, который мы создаем позже.
45. Извлекаем значения a, b и step из соответствующих ячеек текущей строки таблицы.
46. Вычисляем ширину интервала, деля разницу b - a на 3. Это деление позволит нам разбить интегрируемый интервал на три части для параллельных вычислений.
47. Создаем массив totalResult для хранения итогового результата. Используем массив, чтобы обеспечить доступ к значению из потоков.
48. Создаем массив из трех потоков, которые будут использоваться для параллельного вычисления интегралов.
49. Запускаем цикл для инициализации и запуска трех потоков.
50. Вычисляем начальную и конечную точку для каждого потока. Для последнего потока (i == 2) конечная точка будет b, чтобы охватить весь интервал.
51. Создаем новый поток, используя лямбда-выражение.
52. Инициализируем переменную partialSum, которая будет хранить частичную сумму интеграла для текущего потока.
53. Запускаем цикл, который проходит по точкам в пределах start и end, увеличивая x на step. На каждой итерации добавляем к partialSum значение, равное минимальному из step и (b - start).
54. Используем блокировку synchronized, чтобы обеспечить потокобезопасность при добавлении частичной суммы к общему результату. Это необходимо, так как несколько потоков могут одновременно пытаться изменить totalResult.
55. Завершаем определение потока и запускаем его.
56. После запуска всех потоков, мы ждем их завершения с помощью метода join(). Это гарантирует, что основной поток не продолжит выполнение до тех пор, пока все потоки не завершатся.
57. Используем SwingUtilities.invokeLater, чтобы обновить GUI в потоке событий Swing.
58. Обновляем ячейку таблицы в столбце 3 текущей строки finalRow значением totalResult[0], которое содержит итоговый результат интегрирования.
59. Обновляем соответствующий объект RecIntegral, чтобы сохранить результат интегрирования.
60. Отправляем результат на сервер, вызывая метод sendResultToServer с параметрами a, b, step и итоговым результатом.
61. Объявление класса IntegralCalculatorThread, который наследует от Thread, что позволяет ему выполняться в отдельном потоке.
62. Объявление переменной a, представляющей нижний предел интегрирования.
63. Объявление переменной b, представляющей верхний предел интегрирования.
64. Объявление переменной step, представляющей шаг интегрирования.
65. Объявление переменной partialSum, которая будет хранить частичную сумму, вычисленную в потоке.
66. Конструктор класса, который инициализирует переменные a, b, step и устанавливает partialSum в 0.
67. Метод, который будет выполняться в отдельном потоке.
68. Объявление переменных start (начальная точка), h (длина текущего шага) и sum (сумма интегрирования, инициализируемая нулем).
69. Цикл, который выполняется до тех пор, пока start меньше b, что позволяет вычислить интеграл по всему заданному интервалу.
70. Вычисление длины текущего шага h, которая равна минимальному значению между заданным шагом и оставшимся интервалом до b.
71. Обновление суммы интеграла с использованием метода трапеций, где h умножается на среднее значение функции в точках start и start + h.
72. Увеличение start на длину шага h, чтобы перейти к следующему участку интервала.
73. Сохранение вычисленной частичной суммы в переменной partialSum.
74. Метод, который возвращает значение partialSum, позволяя другим частям программы получить результат вычислений, выполненных в потоке.
75. Это объявление метода sendResultToServer, который принимает четыре параметра: a, b, step и result. Эти параметры представляют собой значения, которые будут отправлены на сервер.
76. Начало блока try, который используется для обработки исключений. В данном случае это необходимо, так как работа с сокетами может привести к различным ошибкам, например, если сервер недоступен.
77. Здесь мы формируем строку data, используя метод String.format.
78. Преобразуем строку data в массив байтов с помощью метода getBytes(). Это необходимо, так как данные для отправки в UDP должны быть в виде массива байтов.
79. Создаем новый объект DatagramPacket, который будет содержать данные для отправки.
80. Используем метод send() объекта socket, чтобы отправить пакет данных на сервер.
81. Начало блока catch, который будет перехватывать любые исключения, возникшие в блоке try.
82. Объявление метода, который будет вызываться при нажатии на кнопку 4. Параметр evt представляет собой объект события, который содержит информацию о событии.
83. Получение модели таблицы jTable1 и приведение её к типу DefaultTableModel. Это позволяет изменять данные в таблице.
84. Установка количества строк в модели таблицы в 0. Это очищает таблицу, удаляя все существующие строки, чтобы подготовить её для добавления новых данных.
85. Начало цикла for-each, который перебирает все объекты RecIntegral в списке integralsList. Каждый объект будет доступен через переменную rec.
86. Добавление новой строки в модель таблицы tModel. В качестве параметров передаются значения, полученные из объекта rec.
87. Получение нижнего предела интегрирования.
88. Получение верхнего предела интегрирования.
89. Получение шага интегрирования.
90. Получение результата интегрирования.
91. Объявление метода, который вызывается при нажатии на кнопку 5.
92. Параметр evt представляет собой объект события, который содержит информацию о событии.
93. Получение модели таблицы jTable1 и приведение её к типу DefaultTableModel.
94. Установка количества строк в модели таблицы в 0, что очищает таблицу, удаляя все существующие строки.
95. Очистка списка integralsList, удаляя все объекты RecIntegral из него.
96. Объявление метода, который вызывается при нажатии на кнопку 7.
97. Создание нового объекта JFileChooser, который позволяет пользователю выбрать файл для сохранения.
98. Открытие диалогового окна для сохранения файла. Если пользователь нажимает "Сохранить", то:
99. Получение выбранного пользователем файла.
100. Вызов метода saveToTextFile, который, вероятно, сохраняет данные в выбранный текстовый файл.
101. Объявление метода, который вызывается при нажатии на кнопку 8.
102. Создание нового объекта JFileChooser для выбора файла.
103. Открытие диалогового окна для выбора файла. Если пользователь нажимает "Открыть", то:
104. Получение выбранного пользователем файла.
105. Вызов метода loadFromBinaryFile, который, вероятно, загружает данные из выбранного бинарного файла.
106. Объявление метода, который вызывается при нажатии на кнопку 9.
107. Создание нового объекта JFileChooser, который позволяет пользователю выбрать файл для сохранения.
108. Открытие диалогового окна для сохранения файла. Если пользователь нажимает "Сохранить", то:
109. Получение выбранного пользователем файла.
110. Вызов метода saveToBinaryFile, который, вероятно, сохраняет данные в выбранный бинарный файл.
111. Объявление метода, который вызывается при нажатии на кнопку 6.
112. Создание нового объекта JFileChooser для выбора файла.
113. Открытие диалогового окна для выбора файла. Если пользователь нажимает "Открыть", то:
114. Получение выбранного пользователем файла.
115. Вызов метода loadFromTextFile, который, вероятно, загружает данные из выбранного текстового файла.
116. Объявление класса InvalidDataException, который наследует от стандартного класса Exception. Это пользовательское исключение, которое может использоваться для обработки ошибок, связанных с некорректными данными.
117. Конструктор класса, который принимает сообщение об ошибке и передает его в конструктор родительского класса Exception.
118. Объявление метода, который сохраняет данные в текстовый файл. Метод принимает объект File в качестве параметра.
119. Создание объекта PrintWriter для записи в указанный файл. Использование конструкции try-with-resources гарантирует, что ресурс будет закрыт автоматически после завершения блока.
120. Запись данных о каждом объекте RecIntegral в файл в формате CSV (значения разделены запятыми).
121. Показ сообщения об успешном сохранении данных в текстовый файл.
122. Обработка исключения IOException, которое может возникнуть при работе с файлами.
123. Показ сообщения об ошибке, если возникла проблема при сохранении файла.
124. Объявление метода, который загружает данные из текстового файла.
125. Создание объекта BufferedReader для чтения из указанного файла. Использование конструкции try-with-resources для автоматического закрытия ресурса.
126. Очистка списка integralsList, чтобы удалить старые данные перед загрузкой новых.
127. Получение модели таблицы jTable1 и приведение её к типу DefaultTableModel.
128. Очистка таблицы, устанавливая количество строк в 0.
129. Объявление переменной для хранения строки, считанной из файла.
130. Чтение строк из файла в цикле до тех пор, пока не достигнут конец файла.
131. Разделение строки на части по запятой, чтобы получить значения нижнего пред ела, верхнего предела, шага и результата.
132. Преобразование первой части строки в значение нижнего предела.
133. Преобразование второй части строки в значение верхнего предела.
134. Преобразование третьей части строки в значение шага.
135. Преобразование четвертой части строки в значение результата.
136. Создание нового объекта RecIntegral с загруженными значениями.
137. Добавление новой строки в таблицу с загруженными значениями.
138. Показ сообщения об успешной загрузке данных из текстового файла.
139. Обработка исключений IOException и CustomException, которые могут возникнуть при загрузке файла.
140. Показ сообщения об ошибке, если возникла проблема при загрузке файла.
141. Объявление метода, который сохраняет данные в бинарный файл.
142. Вывод в консоль текущего состояния списка integralsList перед сохранением.
143. Проверка, пуст ли список integralsList или равен null.
144. Показ сообщения об ошибке, если нет данных для сохранения, и выход из метода.
145. Создание объекта FileOutputStream для записи в указанный бинарный файл.
146. Создание объекта ObjectOutputStream для сериализации объектов.
147. Сериализация и запись списка integralsList в бинарный файл.
148. Показ сообщения об успешном сохранении данных в бинарный файл.
149. Обработка исключения IOException, которое может возникнуть при работе с файлами.
150. Объявление метода loadFromBinaryFile, который принимает объект File в качестве параметра. Этот метод предназначен для загрузки данных из двоичного файла.
151. Проверка, существует ли файл и не является ли он пустым. Если файл не существует или его длина равна нулю, выполняется следующий блок кода.
152. Если файл не существует или пуст, отображается диалоговое окно с сообщением об ошибке, и выполнение метода прекращается с помощью return.
153. Открывается блок try-with-resources, который автоматически закроет ресурсы после завершения блока. Создается объект ObjectInputStream, который используется для десериализации объектов из файла. FileInputStream открывает поток для чтения из указанного файла.
154. Десериализуем только integralsList
155. Десериализация объекта из файла. Ожидается, что в файле находится объект типа ArrayList<RecIntegral>, который присваивается переменной integralsList.
156. Получение модели таблицы DefaultTableModel, связанной с jTable1. Это позволяет изменять содержимое таблицы.
157. Установка количества строк в модели таблицы в ноль, что очищает таблицу.
158. Для каждого объекта rec добавляется новая строка в модель таблицы. В строку добавляются значения, полученные с помощью методов getLowerLimit(), getUpperLimit(), getStep() и getResult().
159. После успешной загрузки данных отображается диалоговое окно с сообщением об успешной операции.
160. Обработка исключений. Если во время выполнения блока try возникнет ошибка ввода-вывода (IOException) или ошибка, связанная с классами (ClassNotFoundException), выполнение перейдет в этот блок.
161. В случае возникновения исключения отображается диалоговое окно с сообщением об ошибке, включая текст ошибки.
162. Объявление метода main, который является точкой входа в программу.
163. Использование EventQueue для запуска кода в потоке обработки событий, что позволяет избежать блокировки пользовательского интерфейса.
164. Внутренний класс Runnable, который создает новый экземпляр NewJFrame и делает его видимым, что запускает графический интерфейс приложения.
165. Объявление класса CustomException, который является пользовательским исключением. Класс наследуется от стандартного класса Exception, что позволяет использовать его в механизме обработки исключений Java. Это означает, что CustomException будет вести себя как обычное исключение, но с возможностью добавления специфической логики или сообщений.
166. Объявление конструктора класса CustomException, который принимает одну строку message в качестве параметра. Этот конструктор будет использоваться для создания экземпляров CustomException с определенным сообщением об ошибке.
167. Вызов конструктора родительского класса Exception с помощью ключевого слова super. Это передает строку message в конструктор Exception, что позволяет установить сообщение об ошибке, которое будет доступно при обработке исключения. Таким образом, когда будет выброшено исключение CustomException, сообщение, переданное в конструктор, будет доступно через метод getMessage().
168. Импортирование интерфейса Serializable из пакета java.io. Это позволяет объектам класса RecIntegral быть сериализованными, что означает, что их состояние может быть сохранено и восстановлено, например, при записи в файл.
169. Объявление класса RecIntegral, который реализует интерфейс Serializable. Это означает, что экземпляры этого класса могут быть сериализованы и десериализованы.
170. Объявление статической константы serialVersionUID, которая используется для контроля совместимости сериализованных объектов. Это значение должно изменяться, если структура класса изменяется, чтобы избежать проблем при десериализации.
171. Объявление четырех приватных переменных экземпляра типа double, которые будут хранить значения нижнего предела (lowerLimit), верхнего предела (upperLimit), шага (step) и результата интегрирования (result).
172. Объявление конструктора класса RecIntegral, который принимает четыре параметра: lowerLimit, upperLimit, step и result. Конструктор может выбрасывать исключение CustomException, если переданные значения не соответствуют заданным условиям.
173. Проверка, находится ли lowerLimit в допустимом диапазоне (от 0.000001 до 1000000). Если значение не соответствует условиям, выбрасывается исключение CustomException с соответствующим сообщением.
174. Аналогичная проверка для upperLimit. Если значение не в диапазоне, выбрасывается исключение CustomException.
175. Проверка для step. Если значение не в диапазоне, выбрасывается исключение CustomException.
176. Проверка, что lowerLimit меньше upperLimit. Если это не так, выбрасывается исключение CustomException.
177. Проверка, что разница между upperLimit и lowerLimit больше, чем step. Если это не так, выбрасывается исключение CustomException.
178. Если все проверки пройдены, значения параметров присваиваются соответствующим переменным экземпляра класса.
179. Метод getLowerLimit, который возвращает значение переменной lowerLimit.
180. Метод getUpperLimit, который возвращает значение переменной upperLimit.
181. Метод getStep, который возвращает значение переменной step.
182. Метод getResult, который возвращает значение переменной result.
183. Метод setResult, который позволяет установить новое значение для переменной result.
184. Объявляем класс Server. Это основной класс, который будет содержать логику сервера.
185. Объявляем метод main, который является точкой входа в программу. Он может выбрасывать исключения, что позволяет нам обрабатывать ошибки, связанные с сетевыми операциями.
186. Создаем экземпляр DatagramSocket, который будет слушать входящие UDP-пакеты на порту 9876.
187. Выводим сообщение в консоль, информируя о том, что сервер запущен и готов принимать данные.
188. Запускаем бесконечный цикл, в котором сервер будет постоянно ожидать входящих пакетов.
189. Создаем массив байтов buffer размером 1025, который будет использоваться для хранения данных, получаемых в пакете.
190. Создаем объект DatagramPacket, который будет использоваться для получения данных. Он инициализируется ранее созданным массивом buffer.
191. Ожидаем получения UDP-пакета. Этот метод блокирует выполнение до тех пор, пока не будет получен пакет. Полученные данные будут помещены в packet.
192. Извлекаем данные из пакета и преобразуем их в строку. Метод getData() возвращает массив байтов, а getLength() указывает, сколько байтов было получено.
193. Проверяем, является ли полученное сообщение строкой "CLIENT\_CONNECTED".
194. Если клиент подключен, создаем строку ответа "SERVER\_ACK", которая будет отправлена обратно клиенту.
195. Преобразуем строку ответа в массив байтов.
196. Создаем новый DatagramPacket для отправки ответа. Используем адрес и порт, откуда пришел исходный пакет, чтобы ответить именно этому клиенту.
197. Отправляем ответ клиенту с помощью метода send().
198. Если полученное сообщение не является уведомлением о подключении, выполняем следующий блок кода.
199. Выводим в консоль сообщение о том, что был получен результат от клиента, а также адрес клиента и само сообщение.

Результат выполнения программы

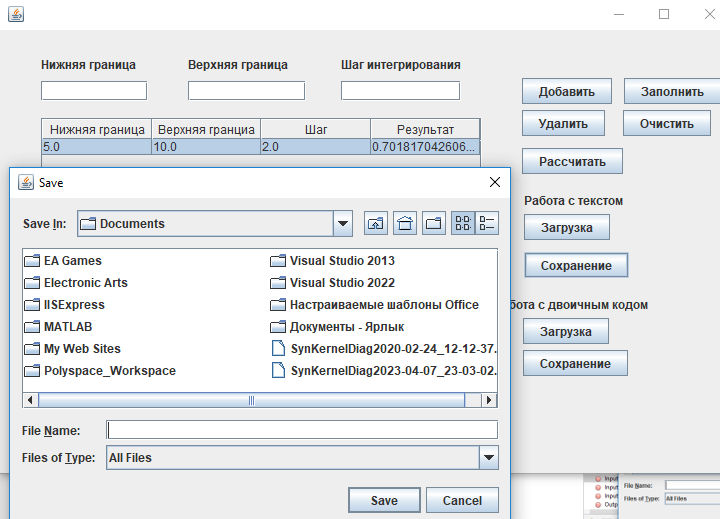


Рисунок 1 — Выбор файла для сохранения

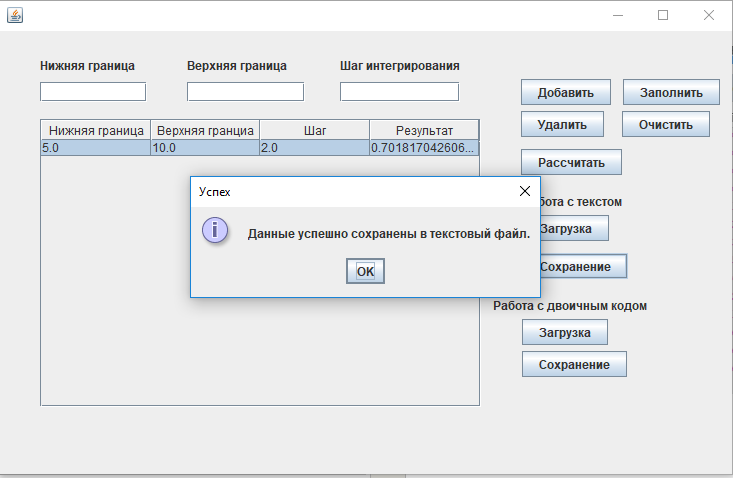


Рисунок 2 — Данные успешно сохранены

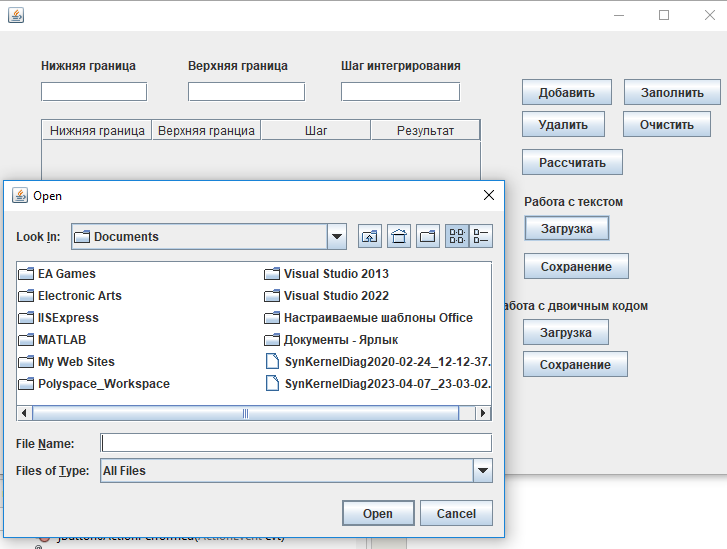


Рисунок 3 — Выбор файла из которого нужно загрузить данные

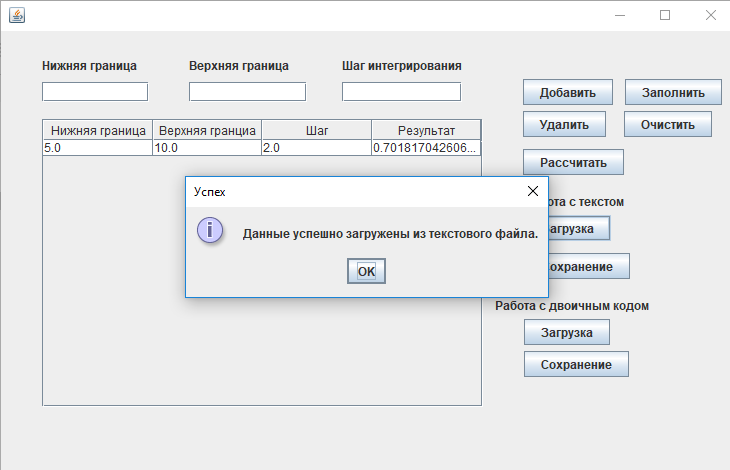


Рисунок 4 — Данные успешно загружены

Вывод

В данной лабораторной работе был получен опыт, чтобы создавать клиент-серверные приложения c использованием стандартных классов Java.